UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE MATEMÁTICA

ISCTE
INSTITUTO UNIVERSITÁRIO DE LISBOA
DEPARTAMENTO DE FINANÇAS

Reinforcement Learning aplicado ao trading

Diogo Miguel Teixeira Estevinho Pires

Mestrado em Matemática Financeira

Dissertação orientada por:
Prof.ª Doutora Diana Elisabeta Aldea Mendes

2020

Resumo

Neste trabalho desenvolve-se um algoritmo de *trading* aplicado ao mercado cambial. O algoritmo é obtido através de *Reinforcement Learning*. Propõe-se um processo de decisão de Markov para modelar o problema do *trading* de um ativo e usa-se o algoritmo *model-free* chamado *Q-Learning* com uma rede neuronal com 2 camadas ocultas de 40 células cada como função aproximação para resolvê-lo. Optou-se por uma função recompensa que calcula a variação percentual do valor da carteira e por um espaço de ações discreto onde as únicas opções são abrir uma posição longa/curta ou não abrir nenhuma posição.

O algoritmo é testado em quatro pares de moedas, EUR/USD, GBP/USD, USD/JYP, USD/CHF, entre abril de 2015 e março de 2020. Além disso, o algoritmo é corrido várias vezes para cada par com o objetivo de testar a consistência do mesmo. Dois valores para o *spread* são utilizados. Os resultados são, em média, consistentes e positivos. Observa-se também um declínio da *performance* com o aumento do *spread*.

Palavras-chave: *Reinforcement Learning*, *Q-Learning*, Redes neuronais, Mercado cambial.

Abstract

In this work, a trading algorithm applied to the foreign exchange market is developed. The algorithm is obtained through Reinforcement Learning. A Markov decision process is proposed to model the problem of trading an asset and a model-free algorithm called Q-Learning is used with a neuronal network with 2 hidden layers of 40 cells each, as an approximation function, to solve it. We opted for a reward function that calculates the percentage variation of the portfolio's value and for a discrete action space where the only options are to open a long/short position or to open no position at all.

The algorithm is tested on four currency pairs, EUR/USD, GBP/USD, USD/JYP, USD/CHF, between April 2015 and March 2020. Besides that, the algorithm is run several times for each pair to test its consistency. Two values for the spread are used. The results are, on average, consistent and positive. There is also a decline in the performance with the increase of the spread.

Keywords: Reinforcement Learning, Q-Learning, Neural Networks, Foreign exchange market.

Índice

Lista de figuras

Lista de tabelas

1. Introdução

Atualmente é relativamente simples e rápido obter informação dos mercados financeiros. Aliada a esta crescente acessibilidade dos mercados, a capacidade dos computadores tem vindo a aumentar consideravelmente permitindo, assim, processar grandes quantidades de informação em muito pouco tempo. Desta forma, o *trading* de forma algorítmica surge com alguma naturalidade, não obstante ser sempre uma tarefa exigente desenvolver um algoritmo eficiente, tendo em conta a volatilidade e instabilidade dos mercados.

Um algoritmo de *Reinforcement Learning (RL)* pode ser interpretado como um algoritmo que aprende a jogar bem um jogo sabendo apenas as respetivas regras. A noção de jogo é formalizada por um processo de decisão de Markov. Nesse sentido, um dos desafios ao aplicar um algoritmo desta categoria é moldar o problema em questão num processo de decisão de Markov de maneira a que a aplicação do algoritmo de *RL* seja natural. Esta transformação do problema inicial nem sempre é trivial e, em alguns contextos, é mais uma arte do que uma ciência. De qualquer das formas, o facto de ser apenas necessário saber as regras do jogo permite-nos aplicar este tipo de algoritmos em muitos contextos de uma forma relativamente simples e prática e, consequentemente, solucionar problemas de alguma complexidade. Na prática, um algoritmo de *RL* aprende por tentativa-erro.

Neste sentido, o propósito deste trabalho é estudar a eficiência de um algoritmo baseado em *RL* no *trading* de um ativo. O objetivo é, no final, obter um programa que a cada momento do tempo nos dê um sinal de compra ou venda sobre o ativo em função de informação extraída da evolução da sua cotação. De uma forma simplificada, usa-se um método chamado *Q-Learning* com uma rede neuronal como função aproximação para aproximar a função Q ótima. A partir desta aproximação deriva-se uma política ótima, que na prática é o critério que nos gera o sinal de compra ou venda sobre o ativo.

No primeiro capítulo é apresentado um suporto teórico sobre *RL*. O objetivo é apenas apresentar os conceitos principais necessários para a compreensão do algoritmo *Q-Learning*. Ainda neste capítulo, introduz-se a variação do *Q-Learning* que resulta da adição de uma rede neuronal como função aproximação.

No segundo capítulo a implementação do algoritmo é abordada com mais detalhe. Esta pode ser dividida em duas fases: a fase de treino e a fase de teste. A fase de treino tem como finalidade obter o critério que gera os sinais de compra e venda. Na fase de teste simula-se o uso do algoritmo criado usando dados históricos dos últimos anos.

No terceiro capítulo são apresentados os resultados dos testes. O algoritmo é aplicado de forma individual aos pares de moedas EUR/USD, GBP/USD, USD/JPY, USD/CHF durante um período de 5 anos.

O trabalho foi escrito assumindo que o leitor tem os conhecimentos básicos sobre redes neuronais. Caso seja necessário, consultar o livro *Neural Networks and Deep Learning (2018)* de Charu C. Aggarwal.

O algoritmo foi implementado em Python e os códigos foram desenvolvidos apenas pelo autor do trabalho.

1.1. Revisão bibliográfica

Durante os últimos anos, foram desenvolvidos vários estudos sobre esta temática. Algumas variações relevantes já foram testadas, como o uso de funções recompensa que têm em consideração a volatilidade do ativo ou até mesmo outros algoritmos de *RL* para além do *Q-Learning*. No entanto, os trabalhos nem sempre são comparáveis de uma forma direta uma vez que são aplicados a diferentes categorias de ativos ou a diferentes horizontes temporais. Nesta secção faz-se uma breve referência a alguns desses estudos.

Huang, Chien-Yi (2018) usa o algoritmo *Q-Learning* com uma rede neuronal como função aproximação para desenvolver um programa de *trading* aplicado ao mercado cambial. O autor usa uma função recompensa que traduz o lucro ou prejuízo em valor absoluto de cada ação. O algoritmo é testado em 12 pares de moedas de forma individual e são usados vários valores para o *spread* avaliando, assim, a *performance* sobre condições menos favoráveis. O autor compara ainda a *performance* quando se usa a técnica intitulada pelo próprio de "*Action Augmentation*" que, simplificando um pouco, é uma técnica para gerar experiências usando a simetria da função recompensa. De um modo geral, os resultados são positivos sendo que com o uso da técnica "*Action Augmentation*" o autor obtém uma *performance* ligeiramente superior.

Zihao Zhang, Stefan Zohren e Stephen Roberts (2019) comparam três algoritmos de *RL*: *Q-Learning*, *Policy Gradients* e *Advantage Actor-Critic*. Nos primeiros dois casos os autores usam um espaço de ações discreto enquanto no último caso usam um espaço contínuo. A função recompensa tem em consideração a volatilidade do ativo em questão visando um comportamento mais conservador por parte do algoritmo. Os algoritmos são testados em contratos futuros sobre várias categorias de ativos. Os resultados são comparados com outras três estratégias clássicas, entre as quais o *buy and hold*. Os autores concluíram que os algoritmos de *RL* têm, no geral, uma *performance* melhor do que as restantes três estratégias, sendo que o *Q-Learning* se destacou pela positiva perante os restantes.

John Moody e Matthew Saffell usam o *Q-Learning* para desenvolver um sistema de *trading* para alocar o capital entre o índice S&P 500 e títulos do tesouro. Os autores propõem uma função recompensa, que batizaram de "*differential Sharpe ratio*", e que visa calcular o impacto das ações sobre o *sharpe ratio*. Obtiveram *performances* superiores em comparação com a estratégia de *buy and hold* sobre o índice S&P 500.

2. Reinforcement Learning

Um processo de decisão de Markov (*Markov Decision Process - MDP*) é uma formalização de um problema de decisão sequencial onde as ações executadas influenciam não só as recompensas numéricas imediatas como também o trajeto. Logo, ao tomar uma decisão, é preciso ter em consideração não só a recompensa imediata, mas também o impacto a médio/longo prazo.

Um *MDP* finito com horizonte infinito pode ser especificado pelo conjunto ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$, $\mathcal{P}$, γ), onde $t = 0, 1, 2, ...$, e onde:

- $\mathcal{S}$ é o conjunto finito dos estados do sistema;
- $\mathcal{A}$ é o conjunto finito das ações;
- $\mathcal{R}$ é o conjunto finito das recompensas;
- $\mathcal{P}: \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ é uma função que nos dá a probabilidade conjunta de transição para um determinado estado e receber uma determinada recompensa, sabendo o estado do sistema no momento anterior e a ação executada. Isto é,

$$\mathcal{P}(s', r, s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a) \tag{2.1}$$

- $\gamma \in [0, 1[$ é o fator de desconto.

Num MDP existe um agente que interage com o sistema em tempo discreto da seguinte forma: a cada momento do tempo t o agente recebe informação sobre o estado do sistema $s_t \in \mathcal{S}$ e, segundo algum critério, escolhe uma ação $a_t \in \mathcal{A}$ para executar. O tempo avança uma unidade, o sistema muda para o estado $s_{t+1} \in \mathcal{S}$ e o agente recebe uma recompensa $r_{t+1} \in \mathcal{R}$ com probabilidade $\mathcal{P}(s_{t+1}, r_{t+1}, s_t, a_t)$. A função $\mathcal{P}$ define por completo a dinâmica do sistema, isto é, sabendo o estado atual e a ação executada, a probabilidade de transição para um determinado estado e receber uma determinada recompensa é independente do percurso efetuado até ao momento. Esta propriedade é conhecida como propriedade de Markov. A figura 2.1 ilustra esta interação.

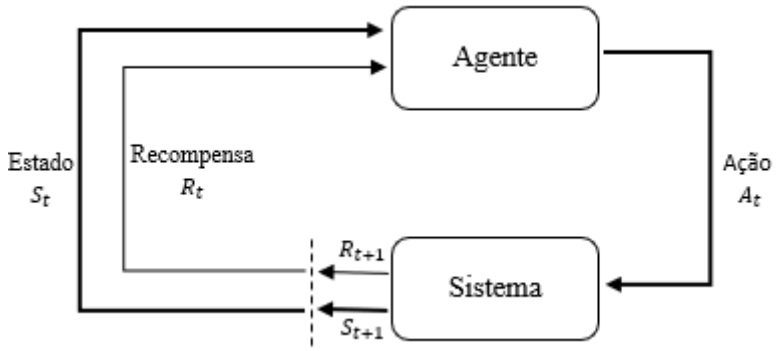

Figura 2.1 - Ilustração da interação do agente com o sistema

Ao critério pelo qual o agente escolhe as ações dá-se o nome de política. Uma política pode ser considerada estacionária ou não estacionária. Diz-se não estacionária se a escolha depende não só do estado como também do tempo em que o sistema se encontra, e diz-se estacionária se a escolha é independente do tempo. Uma política pode ser ainda categorizada como determinística ou estocástica. Uma política estacionária determinística é uma função que a cada estado atribui uma ação, enquanto uma política estacionária estocástica é uma função que a cada par (s, a) atribui a probabilidade de o agente escolher a ação a no estado s. Neste trabalho assume-se que as políticas que o agente pode seguir são estacionárias determinísticas. Para o conjunto das políticas usa-se a notação Π.

Tendo-se já introduzido o conceito de política, pode dizer-se que resolver este *MDP* é encontrar a melhor política possível. À medida que percorre o sistema, o agente vai acumulando recompensas que, de certa forma, refletem o valor do seu percurso. Tendo como objetivo acumular mais e melhores recompensas, a melhor ação num determinado momento é aquela que é capaz de gerar um maior valor em recompensas futuras. Formalmente, o objetivo, ao escolher uma ação no momento t, é maximizar a soma descontada das recompensas futuras:

$$G_t := R_{t+1} + \gamma \times R_{t+2} + \gamma^2 \times R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

O fator de desconto tem uma interpretação pertinente. O seu valor traduz a importância que se dá a recompensas futuras. Para $\gamma = 0$ apenas a recompensa imediata é relevante e à medida que se aumenta o seu valor dá-se cada vez mais relevo às recompensas futuras. Note-se que $\gamma \in [0, 1[$, logo, quanto mais distante do presente estiver a recompensa, menos relevância esta tem.

Uma vez que uma política é o critério de escolha das ações, ela obviamente influencia as recompensas recebidas. Desta forma, existe uma relação entre cada política e as recompensas futuras que serão geradas a partir de cada estado. Esta relação é medida através das funções valor que são definidas a seguir.

Definição (funções valor). Seja π uma política. Chama-se função V para a política π à função que a cada $s \in \mathcal{S}$ atribui o valor $E_\pi[G_t | S_t = s]$ e escreve-se $v_\pi(.)$. Chama-se função Q para a política π à função que a cada par $(s, a) \in \mathcal{S} \times \mathcal{A}$ atribui o valor $E_\pi[G_t | S_t = s,\ A_t = a]$ e escreve-se $q_\pi(.,.)$.

Sobre esta definição impõe-se fazer as seguintes considerações:

- A notação $E_\pi[.]$ representa o valor esperado considerando que as ações são escolhidas seguindo a política π;
- Nem a função V nem a função Q dependem de t, portanto a notação não é ambígua.

Fixada uma política, estas duas funções dão-nos uma noção de qualidade para cada estado e para cada par estado-ação. A função V para a política π diz-nos quão bom é um determinado estado se seguirmos π no futuro. E a função Q para a política π diz-nos quão bom é um determinado estado se escolhermos a ação a nesse mesmo estado e seguirmos π no futuro.

Uma propriedade fundamental da função Q é que ela satisfaz uma relação conhecida como equação de Bellman. Usando a igualdade $G_t = R_{t+1} + \gamma \times G_{t+1}$, vem que:

$$
\begin{aligned}
q_\pi(s,a) &= E_\pi[G_t | S_t = s, A_t = a] \\
&= E_\pi[R_{t+1} + \gamma \times G_{t+1} | S_t = s, A_t = a] \\
&= \sum_{s',r} \mathcal{P}(s', r, s, a) \times [r + \gamma \times E_\pi[G_{t+1} | S_{t+1} = s'\,]\,] \\
&= \sum_{s',r} \mathcal{P}(s', r, s, a) \times [r + \gamma \times v_\pi(s')] \qquad (2.3) \\
&= \sum_{s',r} \mathcal{P}(s', r, s, a) \times [r + \gamma \times q_\pi(s', \pi(s'))] \\
&= E_\pi\big[R_{t+1} + \gamma \times q_\pi\big(S_{t+1}, \pi(S_{t+1})\big) \,\big|\, S_t = s,\ A_t = a\,\big]
\end{aligned}
$$

Ou seja, para uma determinada política π, a função Q pode ser escrita da seguinte forma:

$$
q_\pi(s,a) = E_\pi\big[R_{t+1} + \gamma \times q_\pi\big(S_{t+1}, \pi(S_{t+1})\big) \,\big|\, S_t = s,\ A_t = a\,\big] \qquad (2.4)
$$

Como já foi referido, a função V mede o valor de cada estado segundo uma determinada política. Desta forma, usando estas funções consegue-se comparar as diversas políticas e, consequentemente, definir o conceito de política ótima.

Definição. Diz-se que a política π_1 é melhor que a política π_2 se $v_{\pi_1}(s) \geq v_{\pi_2}(s)$ para todo o $s \in \mathcal{S}$. Diz-se que uma política é uma política ótima se for melhor que todas as outras.

Definição. À função que a cada $s \in \mathcal{S}$ atribui o valor $\max_{\pi \in \Pi} v_\pi(s)$ dá-se o nome de função V ótima e escreve-se $v_*(.)$. À função que a cada par $(s, a) \in \mathcal{S} \times \mathcal{A}$ atribui o valor $\max_{\pi \in \Pi} q_\pi(s, a)$ dá-se o nome de função Q ótima e escreve-se $q_*(.,.)$.

Num MDP finito existe sempre uma política ótima e, caso exista mais do que uma, elas partilham as mesmas funções V e Q. Para além disso, quando se trata de uma política ótima a função V é v_* e a função Q é q_* (ver referência bibliográfica [10], secção 3.6). Assim, para as funções V e Q associadas a uma política ótima usa-se a notação v_* e q_*, respetivamente.

Além do mais, para uma política ótima π^*, mostra-se que a equação de Bellman para a função Q pode ser escrita de uma forma ligeiramente diferente:

$$q_*(s, a) = E_{\pi^*}\left[R_{t+1} + \gamma \times \max_{a' \in \mathcal{A}} q_*(S_{t+1}, a') \mid S_t = s,\ A_t = a\right] \tag{2.5}$$

Neste caso, dá-se o nome de equação de Bellman ótima.

Uma vez conhecida a função Q ótima, uma política ótima fica definida pela igualdade:

$$\pi(s) := arg\max_{a \in \mathcal{A}} q_*(s, a), \qquad \forall s \in \mathcal{S} \tag{2.6}$$

Assim, resolver diretamente a equação (2.5) é um caminho possível para obter a política ótima. Contudo, na prática nem sempre é possível ou eficiente fazê-lo. Para contextos mais complexos ou simplesmente excessivamente grandes, torna-se demasiado pesado computacionalmente. Além disso, é necessário conhecer a dinâmica do sistema, ou seja, a função $\mathcal{P}$. Quando não se tem conhecimento da função $\mathcal{P}$ pode-se optar por métodos que visam aproximar a função Q ótima através da equação de Bellman ótima. Neste trabalho usa-se o algoritmo de *RL* chamado *Q-Learning* com uma rede neuronal como função aproximação. Na próxima secção será abordado este tipo de algoritmo.

2.1. Q-Learning

O objetivo do algoritmo *Q-Learning* é gerar, através da equação de Bellman ótima, uma aproximação sucessiva dos valores da função Q ótima. Para isso o agente tem de interagir com o sistema observando as consequências das suas ações em forma de recompensa numérica.

A versão básica ou clássica deste algoritmo usa uma tabela para guardar os valores das aproximações da função Q ótima. Considere-se uma tabela com um número de linhas igual ao número de estados e um número de colunas igual ao número de ações. Desta forma, para cada par $(s, a) \in \mathcal{S} \times \mathcal{A}$ existe uma entrada na tabela. Para o valor correspondente ao par (s, a) usa-se a notação $h(s, a)$. No início, os valores da tabela são todos 0.

Suponhamos que no momento t o agente observa o estado s_t e escolhe, de alguma forma a definir, a ação a_t. O sistema move-se para o estado s_{t+1} e o agente recebe a recompensa r_{t+1}, formando assim a experiência ($s_t, a_t, s_{t+1}, r_{t+1}$). Antes de escolher uma nova ação a_{t+1} e formar uma nova experiência, atualiza-se o valor de $h(s_t, a_t)$ da seguinte forma:

$$h^{novo}(s_t, a_t) \leftarrow (1-\alpha) \times h(s_t, a_t) + \alpha \times (r_{t+1} + \gamma \times \max_{a' \in \mathcal{A}} h(s_{t+1}, a'))$$

onde $\alpha \in [0,1]$ é um valor fixado à *priori* a que se dá o nome de taxa de aprendizagem. Este processo repete-se a cada momento do tempo. Durante este procedimento, diz-se que o agente está a aprender/treinar.

Estrutura do algoritmo Q-Learning

1. Inicializa-se a tabela
2. A cada momento do tempo:
   a) Observa-se o estado
   b) Escolhe-se e executa-se a ação
   c) Observa-se o novo estado e a recompensa
   d) Atualiza-se a tabela

A convergência deste método é assegurada desde que sejam satisfeitas algumas condições (ver referência bibliográfica [10] e [12]). Essas condições implicam que, para cada estado, cada uma das ações seja executada um número considerável de vezes. Logo, este método não é eficiente se o número de estados e de ações não for suficientemente pequeno. Para ultrapassar este problema, pode-se usar uma função paramétrica Y para estimar os valores da função Q ótima em detrimento do método que se acabou de abordar.

Q-Learning com uma rede neuronal como função aproximação

As redes neuronais são conhecidas pela sua capacidade de processar relações lineares e não lineares e por terem uma boa capacidade de generalização. Neste contexto, diz-se que uma função tem capacidade de generalização se for capaz de gerar uma boa estimativa para qualquer estado, desde que este seja relativamente parecido aos que o agente visitou durante o processo de aprendizagem. Nesse sentido, neste trabalho usa-se uma rede neuronal como função aproximação.

A ideia é agora ir atualizando os parâmetros da rede à medida que o agente percorre o sistema de forma a obter uma função que, quando o *input* é um determinado $s \in \mathcal{S}$, o *output* seja um vetor com as aproximações dos respetivos valores de função Q ótima. Logo, a rede neuronal usada tem uma camada

de *input* de dimensão igual à dos estados e uma camada de *output* de dimensão igual ao número de ações (ver Figura 2.2).

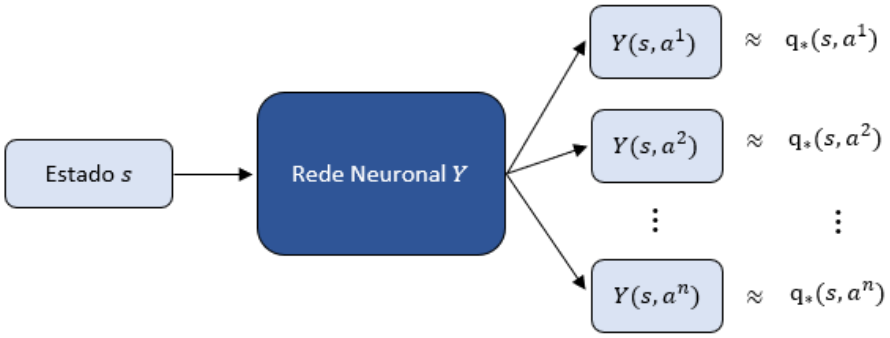

Figura 2.2 - Esquema ilustrativo da rede neuronal principal onde a^k para $1 \leq k \leq n$ são as ações

Antes de se iniciar o mecanismo de aprendizagem, a rede é iniciada com parâmetros aleatórios e é criada uma memória de tamanho k que vai servir para guardar as experiências. O agente começa a interagir com o sistema e as experiências que vai adquirindo são guardadas na memória até esta atingir a sua capacidade máxima. A partir do momento em que a memória é preenchida começa-se a atualizar os parâmetros da função aproximação. No momento em que se adiciona a experiência número k, seleciona-se uma amostra aleatória de experiências da memória e, de seguida, usa-se essa amostra para atualizar os parâmetros da rede através de um algoritmo de gradiente descendente estocástico. O facto de a escolha ser aleatória é importante para não haver correlação entre as experiências.

Para se atualizar a rede é necessário definir os *outputs* desejáveis para cada um dos *inputs*. Considere-se a experiência (s, a, s', r). Neste caso, o *input* é s e o *output* desejável é um vetor que se define da seguinte forma:

- Para a entrada correspondente à ação a o valor é: $r + \gamma \times \max_{a' \in \mathcal{A}} Y(s', a')$;
- Para as restantes entradas o valor é simplesmente $Y(s, a')$, onde a' é a ação em questão. Este valor é calculado através da rede neuronal, introduzindo s como *input*.

No momento seguinte, o agente gera uma nova experiência e, como a memória já está cheia, substitui-se a experiência mais antiga por esta mais recente. Uma amostra de experiências da memória é novamente escolhida de forma aleatória e atualiza-se os parâmetros da rede conforme já foi especificado.

Para evitar alguns problemas de instabilidade que possam surgir enquanto se atualizam os parâmetros da função aproximação, usa-se uma segunda rede neuronal Y' para calcular o valor $\max_{a' \in \mathcal{A}} Y(\,.\,, a')$ de cada um dos *outputs* desejáveis relativos à amostra selecionada. Esta segunda rede é criada no início da aprendizagem como uma cópia da rede principal. Contudo, é atualizada apenas a cada *N* iterações. Esta atualização faz-se copiando os parâmetros da rede principal. Assim, para a experiência (s, a, s', r) o valor da entrada do *output* desejável relativa à ação a passa a ser: $r + \gamma \times \max_{a' \in \mathcal{A}} Y'(s', a')$.

Uma questão importante, e que ainda falta esclarecer, é a de como o agente escolhe as ações a executar durante o processo de aprendizagem. Por um lado, é importante que o agente experimente diferentes ações para cada estado, para que consiga avaliar o impacto de cada uma, isto é, é importante haver uma componente aleatória na escolha de ações durante a aprendizagem, mas, por outro lado, este procedimento pode tornar-se ineficiente para um conjunto de estados e/ou de ações demasiado grande. Nesse sentido, é necessário arranjar um equilíbrio. Neste trabalho usa-se uma estratégia ϵ-*greedy*, ou seja, a cada momento do tempo o agente escolhe a ação de forma aleatória com probabilidade ϵ, e escolhe a ação cuja estimativa para o valor da função Q ótima dada pela rede neuronal principal é maior, com probabilidade $1-\epsilon$. No início, define-se $\epsilon = 1$ e, a cada iteração, reduz-se o seu valor. Desta forma, a probabilidade de o agente executar ações aleatórias vai diminuindo ao longo do tempo.

No final deste processo, a rede neuronal principal deverá gerar uma boa aproximação da função Q ótima e, portanto, deriva-se a política ótima a partir da equação (2.6). Desta forma, a política ótima fica definida pela igualdade:

$$\pi(s) := arg\max_{a \in \mathcal{A}} Y(s,a)\,, \qquad \forall s \in \mathcal{S} \tag{2.7}$$

Estrutura do algoritmo Q-Learning com uma rede neuronal como função aproximação

1. Inicializa-se as duas redes neuronais
2. Inicializa-se a memória
3. A cada momento do tempo:
   a) Observa-se o estado
   b) Escolhe-se e executa-se a ação
   c) Observa-se o novo estado e a recompensa
   d) Atualiza-se a memória
   e) Se a memória tiver cheia:
      i. Seleciona-se uma amostra de experiências da memória
      ii. Atualiza-se os parâmetros da rede neuronal principal com a amostra selecionada
      iii. Atualiza-se a rede secundária, se for o caso

Em suma, resolver um *MDP* traduz-se em encontrar a política ótima que, por sua vez, tendo em consideração a igualdade (2.6), se resume a encontrar a função Q ótima. Para aproximar a função Q ótima pode-se usar o algoritmo *Q-Learning* com uma rede neuronal como função aproximação. Após este método iterativo, a rede neuronal serve como aproximação à função Q ótima. Assim, deduz-se a política ótima através da rede neuronal pela equação (2.7).

3. Implementação

Para se poder implementar o método iterativo do final do capítulo anterior é ainda necessário abordar a questão de como são geradas as experiências para treinar a rede neuronal principal. Por outras palavras, é ainda necessário especificar quais são os estados que o agente visita durante o treino, como são geradas as recompensas e como se define a transição de um estado para o outro. Uma vez feitas essas especificações, já é possível implementar o algoritmo e, consequentemente, obter a política ótima. Com a política ótima definida, testa-se o algoritmo.

Nesse sentido, nas primeiras três secções deste capítulo são definidos conceitos que dizem respeito e que serão fundamentais tanto na fase de treino como na fase de teste. São estes o conjunto das ações, o conceito de posição e as características das redes neuronais usadas. A secção 3.4 diz respeito à fase de treino e a secção 3.5 é relativa à fase de teste.

3.1. Espaço das ações

O mercado cambial é o local onde são transacionadas as divisas. Cada interveniente pode abrir uma posição longa ou uma posição curta sobre um determinado par de moedas. Abrir uma posição num par de moedas corresponde a criar dívida numa das moedas para adquirir a outra moeda. No caso de uma posição longa, cria-se uma dívida na moeda cotada para adquirir a moeda base, e no caso de uma posição curta cria-se dívida na moeda base para adquirir a moeda cotada.

Assim, e uma vez que o algoritmo se aplica apenas a um ativo, considerou-se um espaço de ações composto por três elementos. Em cada momento do tempo, existem 3 ações possíveis, 0, 1 e 2, e cujo significado depende do facto de já existir ou não uma posição aberta.

A ação 0 corresponde a:

- Fechar a posição, se houver alguma aberta;
- Não fazer nada, se não houver nenhuma posição aberta.

A ação 1 corresponde a:

- Não fazer nada se já houver uma posição longa aberta;
- Abrir uma posição longa se não houver nenhuma posição aberta;
- Fechar a posição curta e abrir uma longa, se houver alguma posição curta aberta.

A ação 2 corresponde a:

- Não fazer nada se já houver uma posição curta aberta;
- Abrir uma posição curta se não houver nenhuma posição aberta;
- Fechar a posição longa e abrir uma curta, se houver alguma posição longa aberta.

3.2. Posição

A posição do sistema reflete, como o próprio nome indica, o tipo de posição aberta no momento. Esta informação é relevante, não só devido à definição de cada ação, como também pelo facto de que o algoritmo vai ter em consideração uma comissão que é aplicada quando se abre uma nova posição.

Naturalmente, em cada momento do tempo existem 3 posições possíveis: -1, 0 e 1.

- A posição 0 significa que não há nenhuma posição aberta;
- A posição 1 significa que há uma posição longa aberta;
- A posição -1 significa que há uma posição curta aberta.

A variação do valor da posição depende apenas da ação executada no momento anterior.

- Se o agente executar a ação 0, a posição do sistema no momento seguinte é a posição 0;
- Se o agente executar a ação 1, a posição do sistema no momento seguinte é a posição 1;
- Se o agente executar a ação 2, a posição do sistema no momento seguinte é a posição -1.

3.3. Rede neuronal

Foram usadas redes neuronais de tipo *feedforward* com duas camadas ocultas com 40 células cada. A função de ativação usada tanto nas camadas ocultas como na camada de *output* foi a ReLu (*rectified linear unit*) com um *treshold* de -1, cuja expressão é:

$$ReLu(x) = \begin{cases} x & , se\ x \geq -1 \\ 0 & , se\ x < -1 \end{cases}$$

A função erro usada foi a função Erro Quadrático Médio e o algoritmo de otimização usado foi o Adam (ver referência bibliográfica [1]) com uma taxa de aprendizagem de 0.0002.

3.4. Fase de treino

Nesta secção define-se a função recompensa e explica-se o mecanismo de construção dos estados que o agente visita durante o treino. Note-se que o processo de escolha das ações já foi definido no final do capítulo anterior.

3.4.1. Processamento

Neste contexto do *trading* não existe propriamente uma maneira correta de definir os estados. Não obstante, é necessário ter em consideração que, uma vez fixada uma política, as decisões do agente são tomadas em função apenas do estado do sistema no momento. Ora, no *trading*, faz sentido considerar que as cotações do ativo do passado são, de alguma forma, relevantes para decisões futuras. Desta forma, é essencial os estados serem construídos de modo a conterem informação não só do presente como também do passado recente. A construção dos estados será feita apenas na secção 3.4.3, no entanto é nesta secção que se aborda o conceito de observação composta que vai servir de base para construir os estados.

Para o efeito, usou-se as séries dos preços *Open*, *Low*, *High* e *Close* do ativo num certo horizonte temporal. Transformou-se cada uma destas séries em retornos logarítmicos e, de seguida, normalizou-se. O processo de normalização faz-se da seguinte forma: para cada uma das séries calcula-se a média μ e o desvio-padrão σ, e a cada elemento dessa mesma série aplica-se a função $g(x) := \frac{x-\mu}{\sigma}$. Quando se treina uma rede neuronal considera-se uma boa prática normalizar os dados uma vez que este procedimento, por norma, aumenta a velocidade de aprendizagem e proporciona uma convergência mais rápida.

Para facilitar a explicação introduz-se a notação y_t^k, que corresponde à entrada número t da série número $k \in \{1,2,3,4\}$. Ao vetor $(y_t^1, y_t^2, y_t^3, y_t^4)$ dá-se o nome de observação diária número t. E ao vetor $(y_t^1, y_t^2, y_t^3, y_t^4, y_{t+1}^1, y_{t+1}^2, y_{t+1}^3, y_{t+1}^4, \dots, y_{t+29}^1, y_{t+29}^2, y_{t+29}^3, y_{t+29}^4)$ dá-se o nome de observação composta número t. Assim, uma observação composta resulta da união de 30 observações diárias seguidas. Logo, são vetores de 120 entradas.

Resumindo, a observação composta de um determinado momento corresponde à união dos últimos 30 retornos logarítmicos normalizados das séries dos preços *Open*, *Low*, *High* e *Close*. Feito isto, tem-se uma sequência de observações compostas, que, como já foi referido, vai ser bastante útil para construir os estados da fase de treino.

3.4.2. Função recompensa

A função recompensa é uma função que gera a recompensa obtida pelo agente num determinado momento durante o treino. É uma das partes mais importantes do algoritmo uma vez que influencia diretamente a aprendizagem. A escolha desta função depende do objetivo que se quer atingir ou da métrica que se quer maximizar. Se se quiser apenas maximizar o retorno final faz sentido usar uma função que dê de certa forma a variação do valor da carteira ou do ativo. Por outro lado, se se pretender ter em consideração o risco associado a determinadas ações em certos momentos, pode usar-se uma função que penalize o agente por tomar ações mais arriscadas, esperando-se, assim, que sejam executadas decisões mais conservadoras.

Neste trabalho definiu-se o valor da função recompensa no momento t como $log\ (1 + r_t)$, onde r_t é a variação percentual do valor da carteira do momento $t - 1$ para t. Uma vez mais, usou-se o retorno logarítmico em detrimento do retorno usual pelo facto de parecer mais natural no contexto de *RL* devido à propriedade aditiva dos logaritmos e tendo em conta as definições das funções V e Q. Após esta

definição, o objetivo é agora calcular a variação percentual do valor da carteira entre dois momentos em função da posição, da ação executada e da cotação do ativo ao longo do tempo.

Antes de se avançar impõe-se fazer as seguintes considerações:

- A moeda de referência é sempre a moeda cotada. Ou seja, no caso do EUR/USD a moeda de referência é o dólar;
- É possível comprar/vender qualquer fração do ativo em questão;
- Sempre que se abre uma nova posição, abre-se com o tamanho máximo que se conseguir;
- A comissão c é uma percentagem do valor da dívida criada quando se abre uma posição;
- No momento em que se abre uma posição, o valor que ficará na conta depois de pagar a comissão tem de ser suficiente para pagar, nesse mesmo momento, a dívida criada.

Perante estas observações, a pergunta que surge é: "qual a quantia máxima de dívida que se pode criar ao abrir uma posição tendo em consideração o montante que se tem na conta?".

Sejam C_t e V_t os preços de compra e de venda (respetivamente) do ativo no momento t.

Particularizando-se para o par EUR/USD, suponhamos que se tem uma conta com M dólares e que se abre uma posição longa criando-se uma divida de N dólares. Desta forma, depois de se pagar o valor da comissão, fica-se com $M - N \times c$ dólares e, portanto, tendo em consideração a última observação, a desigualdade $N \leq M - N \times c$ tem de se verificar. Logo, N pode ser no máximo $\frac{M}{(1+c)}$. E, nesse caso, fica-se com $M - N \times c = M - \frac{M \times c}{(1+c)} = \frac{M}{(1+c)}$ dólares na conta.

No caso das posições curtas o raciocínio é ligeiramente diferente uma vez que a dívida é em euros. Supondo que no momento t se cria uma dívida de n euros, o valor da comissão será $n \times c$ euros e, portanto, será necessário gastar-se $n \times c \times C_t$ dólares para pagar esta comissão. A conta ficará com $M - n \times c \times C_t$ dólares e, desta forma, a desigualdade $n \leq \frac{M - n \times c \times C_t}{C_t}$ terá de se verificar. Logo, n pode ser no máximo $\frac{M}{(1+c) \times C_t}$. E, nesse caso, a conta ficará com $M - n \times c \times C_t = \frac{M}{(1+c)}$ dólares.

Resumindo, se se tiver M dólares na conta, para se abrir uma posição longa cria-se uma dívida de $\frac{M}{(1+c)}$ dólares para comprar $\frac{M}{(1+c) \times C_t}$ euros. E para se abrir uma posição curta, cria-se uma dívida de $\frac{M}{(1+c) \times C_t}$ euros para adquirir $\frac{M \times V_t}{(1+c) \times C_t}$ dólares. Em ambos os casos, fica-se com $\frac{M}{(1+c)}$ dólares na conta no momento em que se abre a posição.

Antes de se especificar a função recompensa faz-se ainda dois raciocínios para deduzir a variação percentual do valor da carteira entre dois momentos seguidos.

Raciocínio 1

Particularizando-se uma vez mais para o par EUR/USD, suponhamos que se tem M dólares na conta, que no momento 0 se abre uma posição longa e que só se fecha no momento T. Seja R_k a

variação percentual do valor da carteira entre o momento 0 e o momento k, e r_k a variação percentual do valor da carteira entre $k-1$ e k.

Então, para $1 \leq k \leq T$,

$$R_k = \left(\frac{M \times V_k}{(1+c) \times C_0} - \frac{M}{(1+c)} + \frac{M}{(1+c)} - M\right) \times \frac{1}{M} = \frac{V_k}{C_0 \times (1+c)} - 1 \,. \tag{3.1}$$

Logo $r_1 = R_1 = \frac{V_1}{C_0 \times (1+c)} - 1$.

E, para $1 < k \leq T$, r_k é tal que

$$(1 + R_{k-1}) \times (1 + r_k) = (1 + R_k) \,. \tag{3.2}$$

Ou seja, $r_k = \frac{V_k}{V_{k-1}} - 1$, para $1 < k \leq T$.

Raciocínio 2

De forma análoga, suponhamos que se tem uma conta com M dólares, que no momento 0 se abre uma posição curta e que só se fecha no momento T. Seja R_k a variação percentual do valor da carteira entre o momento 0 e o momento k, e r_k a variação percentual do valor da carteira entre $k-1$ e k.

Então, para $1 \leq k \leq T$,

$$R_k = \left(\frac{M \times V_0}{(1+c) \times C_0} - \frac{M \times C_k}{(1+c) \times C_0} + \frac{M}{(1+c)} - M\right) \times \frac{1}{M} = \frac{V_0 - C_k + C_0}{C_0 \times (1+c)} - 1 \,. \tag{3.3}$$

Logo $r_1 = R_1 = \frac{V_0 - C_1 + C_0}{C_0 \times (1+c)} - 1$.

E, para $1 < k \leq T$, r_k é tal que

$$(1 + R_{k-1}) \times (1 + r_k) = (1 + R_k) \,. \tag{3.4}$$

Ou seja, $r_k = \frac{C_{k-1} - C_k}{C_0 + V_0 - C_{k-1}}$, para $1 < k \leq T$.

Observe-se que, enquanto a posição curta está aberta, tem-se uma dívida de $\frac{M}{(1+c) \times C_0}$ euros. Desta forma, se se quisesse fechar a posição no momento k seria necessário ter-se $\frac{M \times C_k}{(1+c) \times C_0}$ dólares e, portanto, enquanto uma posição curta está aberta, verifica-se a cada momento se este valor não ultrapassa o montante total disponível. Concretizando, verifica-se se

$$\frac{M\times V_0}{(1+c)\times C_0}+\frac{M}{(1+c)} > \frac{M\times C_k}{(1+c)\times C_0} \Leftrightarrow V_0 + C_0 > C_k \,. \quad (3.5)$$

Caso esta desigualdade não se verifique em algum momento, assume-se que se foi à falência. Desta forma, o denominador dos r_k no raciocínio 2 é sempre positivo. No caso das posições longas não existe este problema uma vez que a dívida é em dólares e o seu valor é igual ao montante existente na conta.

Função recompensa

Tendo em consideração o raciocínio 1 e 2, e interpretando p como a posição do sistema no momento t e a como a ação executada no momento t, a variação percentual r do valor da carteira do momento t para $t+1$ é dada pelo seguinte esquema:

- Se $p = 0$ e:
  - Se $a = 0$, então $r = 0$
  - Se $a = 1$, então $r = \frac{V_{t+1}}{C_t\times(1+c)} - 1$
  - Se $a = 2$, então $r = \frac{V_t - C_{t+1} + C_t}{C_t\times(1+c)} - 1$
- Se $p = 1$ e:
  - Se $a = 0$, então $r = 0$
  - Se $a = 1$, então $r = \frac{V_{t+1}}{V_t} - 1$
  - Se $a = 2$, então $r = \frac{V_t - C_{t+1} + C_t}{C_t\times(1+c)} - 1$
- Se $p = -1$ e:
  - Se $a = 0$, então $r = 0$
  - Se $a = 1$, então $r = \frac{V_{t+1}}{C_t\times(1+c)} - 1$
  - Se $a = 2$, então $r = \frac{C_t - C_{t+1}}{C_d + V_d - C_t}$, onde d é o momento em que se abriu a posição curta.

3.4.3. Treino

Nesta secção irá juntar-se as ideias desenvolvidas ao longo deste capítulo para explicar como são geradas as experiências durante o treino.

Os estados que o agente visita durante a fase de treino são vetores de 121 entradas, onde a primeira entrada corresponde ao valor da posição e as restantes entradas correspondem aos valores de uma determinada observação composta. Para esta construção usa-se o termo "união".

No primeiro estado da fase de treino define-se que a posição é 0, logo o primeiro estado é a união do valor 0 com a primeira observação composta. Depois de observar o primeiro estado, o agente escolhe uma ação a executar de acordo com o método explicado no final do capítulo anterior. O sistema move-se para o segundo estado que resulta da união do valor da posição, cujo cálculo é especificado na secção 3.2, com a segunda observação composta. A recompensa obtida é dada pela função recompensa. Tem-se, assim, a primeira experiência da fase de treino. No momento seguinte, o agente escolhe uma nova ação, o sistema move-se para o estado que resulta da união da posição do sistema com a terceira observação composta e o agente recebe uma recompensa dada pela função recompensa, formando, assim, a segunda experiência.

Este processo repete-se até se chegar à última observação composta, ou seja, ao último estado, completando, desta forma, o que se chama de episódio. O sistema volta ao estado inicial e o processo repete-se, completando o segundo episódio. Neste trabalho foram usados 20 episódios. Note-se que nem as redes nem a memória sofrem alterações cada vez que o agente começa um novo episódio. No fundo é um mecanismo para que o agente tenha oportunidade de explorar melhor o sistema, ou seja, para gerar mais experiências.

Juntando o processo descrito na secção 2.1 com o processo descrito neste capítulo, tem-se tudo o que é necessário para implementar a fase de treino do algoritmo. Por fim, falta especificar alguns dos parâmetros usados: para o facto de desconto γ usou-se o valor 0.95, o tamanho usado para a memória foi de 5000 e o tamanho das amostras de 400. A atualização da rede neuronal secundária foi feita a cada 500 iterações.

3.5. Fase de teste

Depois da fase de treino, obtém-se a política definida pela igualdade:

$$\pi(s) := arg\max_{a \in \mathcal{A}} Y(s,a)\,, \qquad \forall s \in \mathcal{S} \tag{3.6}$$

onde $Y(s,a)$ é a aproximação do valor da função Q ótima relativo ao par (s,a) e obtida pela rede principal. O objetivo agora é simular de forma realista o uso do algoritmo. Deste modo, a ideia é deixar o agente percorrer o sistema usando esta política e avaliar, de alguma forma, o seu percurso.

Para isso, primeiro constrói-se, usando um horizonte temporal diferente do da fase de treino, uma sequência de observações compostas da mesma forma que na fase de treino, mas com a diferença de que no processo de normalização os valores de μ e de σ de cada uma das quatro séries são os mesmos usados para construir as observações compostas usadas na aprendizagem. Se esta modificação não fosse feita, estar-se-ia a usar informação do futuro durante a simulação, o que não é realista.

Uma vez mais, define-se o primeiro estado como a união da posição 0 com a primeira observação composta da fase de teste. O agente observa este estado e escolhe uma ação segundo a política definida

pela equação (3.6). De seguida, o sistema move-se para o segundo estado, que corresponde à união da nova posição, que é calculada conforme especificado na secção 3.2, com a segunda observação composta. Este processo repete-se até se chegar ao último estado, terminando, assim, a fase de teste.

Agora é necessário avaliar este percurso. Cada vez que o agente toma uma decisão e o sistema se move para um novo estado, o valor da carteira sofre uma variação percentual r_t. Neste caso, esta variação é dada pelo esquema do final da secção 3.4.2. Assim, no final do percurso de teste tem-se uma sequência finita dos retornos entre cada momento e que é usada para avaliar a *performance* do algoritmo durante o teste.

Essa avaliação é feita segundo três métricas:

- O retorno final, que é dado pela fórmula $\prod_t(1 + r_t)$;
- Percentagem de decisões que geraram um retorno positivo;
- Percentagem de decisões que geraram um retorno negativo.

4. Teste

Neste capítulo são apresentados os resultados obtidos. O algoritmo foi testado para os pares de moedas EUR/USD, GBP/USD, USD/JPY, USD/CHF durante um período de 5 anos. O intervalo entre cada momento do tempo corresponde a 1 dia útil, ou seja, o algoritmo executa apenas uma ação por dia útil.

A escolha do horizonte temporal de treino e de teste é um pouco arbitrária, no entanto, assumindo-se que a dinâmica do ativo pode mudar ao longo do tempo, não convém usar intervalos muito grandes. Por outro lado, se estes intervalos forem muito curtos faz com que a *performance* fique demasiado dependente do horizonte temporal usado. Nesse sentido, utilizou-se dados históricos de 3 anos consecutivos para treinar e testou-se nos 6 meses seguintes. De forma a completar 5 anos consecutivos de teste, o algoritmo foi testado em 10 pares treino-teste.

Tabela 4.1 - Horizontes temporais relativos a cada par treino-teste

	Treino	Teste
Par 1	De 2016-10-01 Até 2019-09-30	De 2019-10-01 Até 2020-03-31
Par 2	De 2016-04-01 Até 2019-03-31	De 2019-04-01 Até 2019-09-31
Par 3	De 2015-10-01 Até 2018-09-30	De 2018-10-01 Até 2019-03-31
Par 4	De 2015-04-01 Até 2018-03-31	De 2018-04-01 Até 2018-09-31
Par 5	De 2014-10-01 Até 2017-09-30	De 2017-10-01 Até 2018-03-31
Par 6	De 2014-04-01 Até 2017-03-31	De 2017-04-01 Até 2017-09-31
Par 7	De 2013-10-01 Até 2016-09-30	De 2016-10-01 Até 2017-03-31
Par 8	De 2013-04-01 Até 2016-03-31	De 2016-04-01 Até 2016-09-31
Par 9	De 2012-10-01 Até 2015-09-30	De 2015-10-01 Até 2016-03-31
Par 10	De 2012-04-01 Até 2015-03-31	De 2015-04-01 Até 2015-09-31

Definiu-se os preços de venda do ativo como os preços da série *Close* e os preços de compra como os preços de venda acrescidos de um *spread* fixo. Para esse *spread* usou-se os valores de 0.0002 e 0.0005 para os pares EUR/USD, GBP/USD, USD/CHF, e os valores 0.02, 0.05 para o par USD/JPY. O valor da comissão usado foi sempre de 0.01%.

Como o algoritmo depende de processos aleatórios em alguns momentos, como por exemplo na escolha dos parâmetros das redes quando estas são criadas e no mecanismo de escolha das ações durante a aprendizagem, a *performance* varia sempre um pouco. Assim, para averiguar a consistência, e uma vez fixado o ativo, um par de treino-teste e um valor para o *spread*, correu-se o algoritmo 20 vezes. Desta forma, por cada combinação ativo-par-*spread*, obtém-se três sequências de 20 elementos cada, onde

cada sequência diz respeito a uma das métricas definidas no final do capítulo anterior. Para a sequência dos retornos finais calculou-se a média, o valor máximo e o mínimo. Para as sequências de percentagens de decisões que geraram um retorno positivo e negativo calculou-se apenas a média, à qual se deu o nome de "W" e "L" respetivamente.

Todas as informações relativas às cotações dos ativos foram obtidas através do *Yahoo Finance*.

4.1. Escolha de parâmetros

Ao longo do algoritmo é necessário definir valores para alguns parâmetros, como por exemplo o tamanho da memória, o número de episódios, o fator de desconto, etc. Fazer esta escolha de forma aleatória não faz muito sentido. Obviamente a escolha nunca é 100% aleatória porque para cada parâmetro existe um intervalo para o qual seja extremamente desajustado considerar valores fora desse conjunto. Ainda assim, é pertinente tentar arranjar algum critério para encontrar uma combinação de parâmetros ótima.

Um critério possível seria definir um intervalo para cada parâmetro e criar uma partição para cada um deles. Desta forma, existe uma combinação finita de parâmetros. E para cada uma dessas combinações podia-se testar o algoritmo e escolher a combinação que gera melhores resultados para um determinado horizonte temporal. No entanto, este procedimento é extremamente demorado dado o número de combinações e o tempo que o próprio algoritmo demora.

Nesse sentido, optou-se por fazer uma análise informal e pouco metódica para várias combinações tendo como *feedback* os resultados gerados pelo algoritmo para os quatro ativos, para ambos os *spreads* e sobre um determinado horizonte temporal de treino-teste. A este intervalo temporal deu-se o nome de Par de validação e definiu-se da seguinte forma:

Tabela 4.2 - Horizonte temporal do Par de validação

	Treino	**Teste**
Par de validação	De 2011-10-01 Até 2014-09-30	De 2014-10-01 Até 2015-03-31

Com os parâmetros escolhidos, os resultados para o Par de validação foram os seguintes:

EUR/USD

Tabela 4.3 - Resultados relativos ao ativo EUR/USD no par de validação (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par de validação	Retorno final	Média	8.76	0.46
		Máximo	23.55	17.7
		Mínimo	-5.75	-12.16
	W		40.19	31.71
	L		34.29	35.27

GBP/USD

Tabela 4.4 - Resultados relativos ao ativo GBP/USD no par de validação (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par de validação	Retorno final	Média	3.48	0.72
		Máximo	13.31	7
		Mínimo	-3.2	-8.07
	W		36.87	35.9
	L		31.99	32.3

USD/JPY

Tabela 4.5 - Resultados relativos ao ativo USD/JPY no par de validação (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,02	0,05
Par de validação	Retorno final	Média	13.35	5.58
		Máximo	30.73	19.54
		Mínimo	-1.38	-9.07
	W		40.47	37.38
	L		31.25	33.2

USD/CHF

Tabela 4.6 - Resultados relativos ao ativo USD/CHF no par de validação (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par de validação	Retorno final	Média	15.54	3.1
		Máximo	39.65	30.65
		Mínimo	-25.83	-22.45
	W		39.34	38.24
	L		31.84	34.8

Pelas tabelas 4.3, 4.4, 4.5 e 4.6, e considerando a média dos retornos finais, pode observar-se que o algoritmo produziu resultados positivos para todas as combinações ativo-*spread*. Verifica-se também que com o aumento do *spread* existe uma diminuição da *performance* de um modo geral. Ou seja, não só existe um decréscimo considerável da média dos retornos finais, como também se verifica uma diminuição da percentagem média de ações que geraram um retorno positivo, um aumento da percentagem média de ações que geraram um retorno negativo e uma redução do retorno final, tanto da melhor como da pior *performance*. Tendo em consideração ambos os *spreads*, verifica-se que os pares USD/JPY e USF/CHF obtiveram os melhores resultados enquanto que o par GBP/USD obteve o pior resultado no Par de validação. Os resultados dos testes são apresentados na próxima secção.

4.2. Resultados

EUR/USD

Tabela 4.7 - Resultados dos testes para o ativo EUR/USD (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par 1	Retorno final	Média	7,71	6,29
		Máximo	20,05	14,39
		Mínimo	-3,69	-0,69
	W		41,67	35,39
	L		30,77	31
Par 2	Retorno final	Média	5.32	3.16
		Máximo	8.8	8.67
		Mínimo	-0.92	-4.75
	W		41.64	35.07
	L		31.91	30.11
Par 3	Retorno final	Média	4.85	4.76
		Máximo	17.31	16.35
		Mínimo	-2.67	-6.65
	W		39.81	40.58
	L		31.55	33.99
Par 4	Retorno final	Média	5.87	6.16
		Máximo	15.42	14.61
		Mínimo	-6.91	-4.66
	W		38.8	38.29
	L		30.73	29.88
Par 5	Retorno final	Média	10.02	6.66
		Máximo	20.02	14.36
		Mínimo	2.41	-2.87
	W		41.4	39.45
	L		28.09	28.79

Tabela 4.8 - Resultados dos testes para o ativo EUR/USD (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par 6	Retorno final	Média	4.88	4.56
		Máximo	13.23	12.55
		Mínimo	-5.81	-10.11
	W		37.42	35.78
	L		31.13	29.8
Par 7	Retorno final	Média	6.8	3.04
		Máximo	12.97	15.16
		Mínimo	-0.92	-3.41
	W		39.33	40.19
	L		29.92	33.67
Par 8	Retorno final	Média	7.25	7.26
		Máximo	15.95	18.73
		Mínimo	0.15	-1.03
	W		38.85	39.73
	L		30.85	31.38
Par 9	Retorno final	Média	4.5	7.08
		Máximo	17.27	19.63
		Mínimo	-2.66	-4.76
	W		37.01	39.77
	L		31.66	30.93
Par 10	Retorno final	Média	4.96	4.77
		Máximo	15.43	15.98
		Mínimo	-7.65	-8.83
	W		37.03	37.58
	L		34.18	32.89

GBP/USD

Tabela 4.9 - Resultados dos testes para o ativo GBP/USD (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par 1	Retorno final	Média	11,07	9,33
		Máximo	29,93	28,98
		Mínimo	-5,86	-8,46
	W		38,45	37,93
	L		33,1	33,25
Par 2	Retorno final	Média	9.97	10.63
		Máximo	15.84	20.71
		Mínimo	4.28	1.36
	W		37.11	39.3
	L		25.58	27.62
Par 3	Retorno final	Média	12.59	13.5
		Máximo	33.16	26.58
		Mínimo	00.98	3.92
	W		40.85	40.5
	L		29.03	27.56
Par 4	Retorno final	Média	9.86	10.77
		Máximo	21.74	25.36
		Mínimo	2.82	3.59
	W		38.8	39.96
	L		27.98	30.12
Par 5	Retorno final	Média	9.25	8.8
		Máximo	19.07	15.35
		Mínimo	-0.2	0.84
	W		40.66	40.39
	L		33.82	33.32

Tabela 4.10 - Resultados dos testes para o ativo GBP/USD (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par 6	Retorno final	Média	8.3	7.27
		Máximo	18.92	15.96
		Mínimo	-2.35	-1.66
	W		40.9	39.18
	L		29.76	32.03
Par 7	Retorno final	Média	5.07	4.82
		Máximo	24.46	20.65
		Mínimo	-9.66	-5.79
	W		36.99	36.48
	L		34.45	33.16
Par 8	Retorno final	Média	5.28	6.77
		Máximo	19.95	29.12
		Mínimo	-9.44	-13.84
	W		39.65	38.65
	L		32.15	31.69
Par 9	Retorno final	Média	10.03	5.49
		Máximo	20.76	18.33
		Mínimo	0.05	-4.6
	W		41.47	37.44
	L		32.36	33.37
Par 10	Retorno final	Média	7.64	6.32
		Máximo	19.47	17.7
		Mínimo	-1.38	-4.54
	W		38.75	38.59
	L		31.56	32.03

USD/JPY

Tabela 4.11 - Resultados dos testes para o ativo USD/JPY (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,02	0,05
Par 1	Retorno final	Média	7,95	9,22
		Máximo	21,4	21,69
		Mínimo	-5,49	-6,04
	W		39,5	36,63
	L		30,04	31,51
Par 2	Retorno final	Média	6.74	4.69
		Máximo	15.55	14.17
		Mínimo	1.05	-4.17
	W		40.55	37.97
	L		32.58	33.48
Par 3	Retorno final	Média	7.31	5.37
		Máximo	15.78	16.68
		Mínimo	2.05	-2.53
	W		40.77	35.27
	L		30.35	30.23
Par 4	Retorno final	Média	4.3	5.37
		Máximo	11.69	11.46
		Mínimo	-4.36	0.75
	W		35.62	35.77
	L		32.25	29.3
Par 5	Retorno final	Média	7.17	6.09
		Máximo	13.69	16.24
		Mínimo	3.05	0.2
	W		38.28	36.99
	L		29.14	29.06

Tabela 4.12 - Resultados dos testes para o ativo USD/JPY (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,02	0,05
Par 6	Retorno final	Média	5.07	6.67
		Máximo	15.32	12.84
		Mínimo	-4.05	0.69
	W		35.78	37.58
	L		31.91	33.4
Par 7	Retorno final	Média	5.4	2.62
		Máximo	14.07	10.56
		Mínimo	-8.89	-6.34
	W		35.78	38
	L		29.92	31.91
Par 8	Retorno final	Média	6.14	9.64
		Máximo	18.95	22.03
		Mínimo	-13.23	-3.41
	W		38.92	36.23
	L		35.46	31.92
Par 9	Retorno final	Média	6.8	4.84
		Máximo	21.75	11.64
		Mínimo	-5.25	-4.1
	W		38.53	38.6
	L		31.05	33.06
Par 10	Retorno final	Média	13.27	9.82
		Máximo	20.48	17.39
		Mínimo	3.9	-0.13
	W		44.72	39.02
	L		31.48	29.18

USD/CHF

Tabela 4.13 - Resultados dos testes para o ativo USD/CHF (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par 1	Retorno final	Média	5,52	5,05
		Máximo	15,63	13,95
		Mínimo	-2,6	-3,39
	W		37,25	35,04
	L		31,16	32,21
Par 2	Retorno final	Média	6.44	3.8
		Máximo	13.37	12
		Mínimo	-7.36	-5.59
	W		40.51	34.76
	L		31.84	30.12
Par 3	Retorno final	Média	5.55	2.64
		Máximo	12	13.08
		Mínimo	-0.35	-2.91
	W		39.96	34.42
	L		30.85	34.38
Par 4	Retorno final	Média	6.17	3.46
		Máximo	12.48	11.38
		Mínimo	-6.04	-4.85
	W		40.5	36.67
	L		30.19	29.73
Par 5	Retorno final	Média	11.17	5.34
		Máximo	20.71	17.75
		Mínimo	5.15	-7.84
	W		42.89	37.89
	L		26.37	29.84

Tabela 4.14 - Resultados dos testes para o ativo USD/CHF (os valores correspondem a percentagens e foram arredondados à 2ª casa decimal).

			spread	
			0,0002	0,0005
Par 6	Retorno final	Média	7.29	4.99
		Máximo	16.83	17.37
		Mínimo	-2.45	-4.06
	W		38.59	33.55
	L		32.54	30.98
Par 7	Retorno final	Média	5.39	4.17
		Máximo	13.16	13.77
		Mínimo	-6.63	-3.33
	W		39.14	39.18
	L		29.53	31.48
Par 8	Retorno final	Média	9.61	10.36
		Máximo	20.04	18.72
		Mínimo	-0.4	-1.17
	W		38.15	38.15
	L		30.58	29.04
Par 9	Retorno final	Média	8.48	7.45
		Máximo	26.59	18.5
		Mínimo	-4.07	-0.94
	W		34.49	41.24
	L		28.18	30.58
Par 10	Retorno final	Média	5.33	3.92
		Máximo	24	20.08
		Mínimo	-9.57	-9.22
	W		36.72	37.58
	L		31.13	33.67

Como se pode constatar pela observação das tabelas numeradas de 4.7 a 4.14, o algoritmo obtém, em média, resultados positivos para todos os casos. Para o *spread* maior os resultados são por norma inferiores, como já era expetável. Ainda assim, existem casos em que a *performance* melhora ligeiramente com o aumento do valor do *spread*. Apesar de ser um cenário pouco intuitivo à *priori*, não deixa de ser válido uma vez que a função recompensa tem em consideração o valor do *spread*. Ou seja, o *spread* influencia a aprendizagem e consequentemente a *performance*. Logo, neste contexto, é natural pensar que com o aumento do *spread* o algoritmo não mude de posição com tanta frequência, o que pode eventualmente ser benéfico.

Avaliando os extremos dos retornos finais, podemos também constatar que, para cada caso, o máximo e o mínimo estão consideravelmente distantes, o que indica que o retorno final pode variar bastante. Além disso, e ao contrário do que seria natural pensar, o aumento do *spread* não implica uma diminuição do retorno máximo e/ou do retorno mínimo. Existe um grande número de casos em pelo menos um destes valores aumenta. Também se constata que, para todas as combinações ativo-par-*spread*, a média das percentagens de retornos positivos é sempre maior que a média de retornos negativos.

Em suma, o algoritmo é volátil a nível de retorno final. Contudo, quando se considera a média das *performances*, este revela-se consistente.

5. Conclusão

Este trabalho serviu para aferir a aplicabilidade de um algoritmo de *RL* no *trading* de pares de moedas. Os mercados financeiros são ambientes bastante atribulados e não estacionários, nos quais a capacidade de generalização de um algoritmo de *RL* é fundamental, portanto, à partida, a obtenção de resultados no mínimo razoáveis, não é óbvia. Para o efeito modulou-se o problema do *trading* de um ativo como um *MDP* e de seguida usou-se o algoritmo *Q-Learning* para o resolver. A função recompensa usada traduz a variação percentual do valor da carteira. Para cada cenário que se considerou, o algoritmo obteve resultados algo dispersos, no entanto, quando se considerou a média dos retornos finais este revelou-se consistente e positivo.

Quanto à metodologia usada, é importante realçar que a escolha dos parâmetros não foi feita de forma metódica nem muito rigorosa, deixando, assim, a dúvida se existe uma combinação que consiga gerar melhores resultados. O tamanho dos horizontes temporais treino-teste também não foi muito explorado.

Para um trabalho futuro, existem algumas modificações que podem ser efetuadas para tornar o programa um pouco mais realista ou simplesmente para averiguar se existe algum aumento a nível de *performance*. Uma variação relevante seria usar uma função recompensa que tivesse em consideração a volatilidade do ativo ou até mesmo o *Sharpe ratio* e comparar, não só o retorno final, como também a volatilidade do algoritmo. Outra variação, esta para tornar o programa mais complexo e versátil, seria permitir que a carteira fosse composta por mais de um ativo em simultâneo e que o algoritmo decidisse a fração do capital a alocar em cada um. Desta forma dar-se-ia muito mais liberdade ao programa.

6. Bibliografia

[1] Aggarwal, C. C. (2018). *Neural Networks and Deep Learning.* Springer.

[2] Busoniu , L., Babuska, R., Bart De Schutter, & Ernst , D. (2010). *Reinforcement learning and dynamic programming using function approximators.* CRC Press.

[3] Carapuço, J. B. (2017). Reinforcement Learning Applied to Forex Trading. Dissertação de mestrado. IST.

[4] Cumming, J. (2015). An Investigation into the Use of Reinforcement Learning Techniques within the Algorithmic Trading Domain. Dissertaçao de mestrado. Imperial College London.

[5] Doloc, C. (2019). *Applications of Computational Intelligence in Data-Driven Trading.* Wiley.

[6] François-Lavet, V., Islam, R., Pineau, J., Henderson, P., & Bellemare, M. G. (2018). *An Introduction to Deep Reinforcement Learning.*

[7] Huang, C.-Y. (2018). Financial Trading as a Game: A Deep Reinforcement Learning Approach.

[8] Moody , J., & Saffell, M. (s.d.). Reinforcement Learning for Trading.

[9] Moody, J., Wu, L., Liao, Y., & Saffell, M. (1997). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting, Volume 17*, 441-470.

[10] Sutton , R. S., & Barto, A. G. (2018). *Reinforcement Learning: an introduction (2ª edition).* The MIT Press.

[11] TAHRI, C. (2019). Reinforcement learning approach for algorithmic trading of bitcoin. Dissertação de mestrado. Univesity of Oklahoma, Graduate College.

[12] Watkins, C. J., & Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, 8, 279-292.

[13] Zhang, Z., Zohren, S., & Roberts, S. (2019). Deep Reinforcement Learning for Trading.